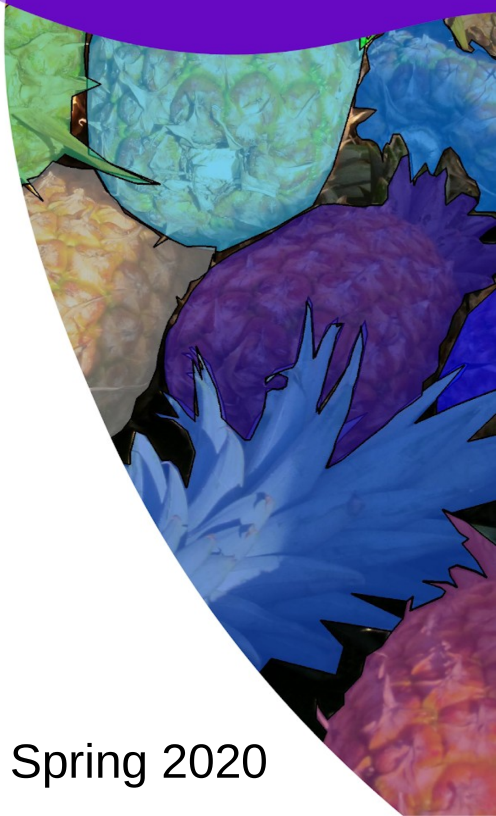
NEW YORK UNIVERSITY

Convolutional Networks

http://bit.ly/DLSP20

Yann LeCun
NYU - Courant Institute & Center for Data Science
Facebook AI Research
http://yann.lecun.com
TAs: Alfredo Canziani, Mark Goldstein

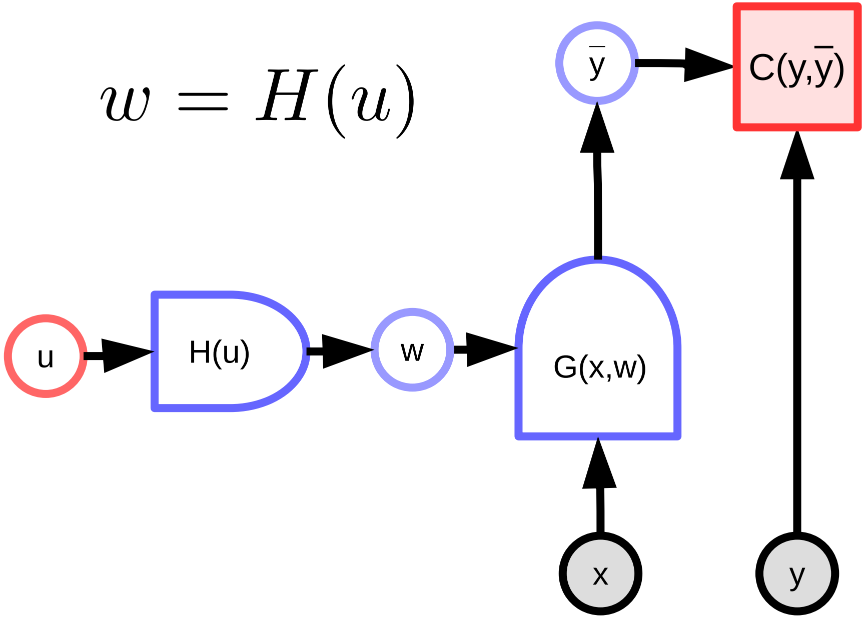


Deep Learning, NYU, Spring 2020

Parameter transformations

- **When the parameter vector is the output of a function**

$$w = H(u)$$



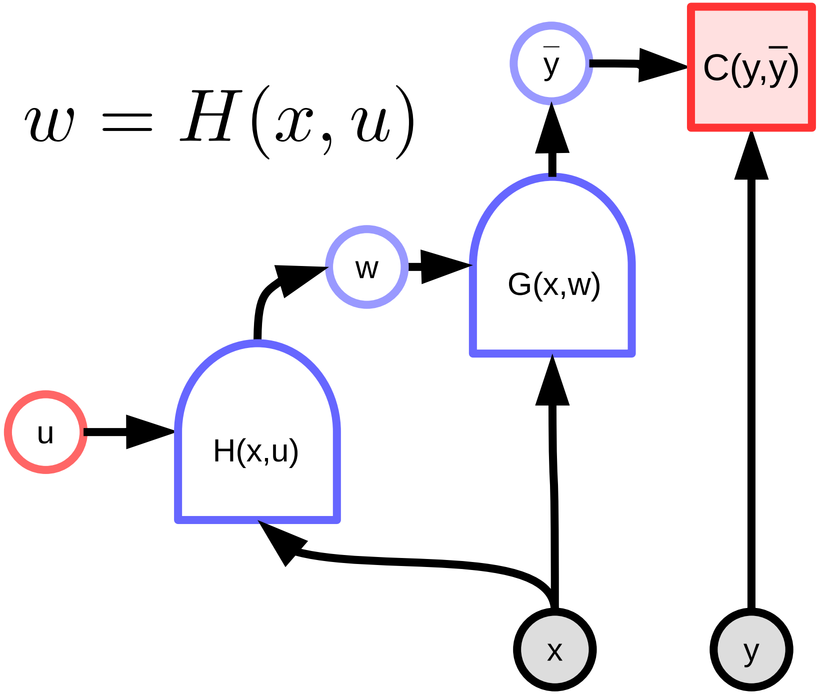
$$u \leftarrow u - \eta \frac{\partial H}{\partial u}^T \frac{\partial C}{\partial w}^T$$

$$w \leftarrow w - \eta \frac{\partial H}{\partial u} \frac{\partial H}{\partial u}^T \frac{\partial C}{\partial w}^T$$

$$[N_w \times N_u]\ [N_u \times N_w]\ [N_w \times 1]$$

“Hypernetwork”

$$w = H(x, u)$$

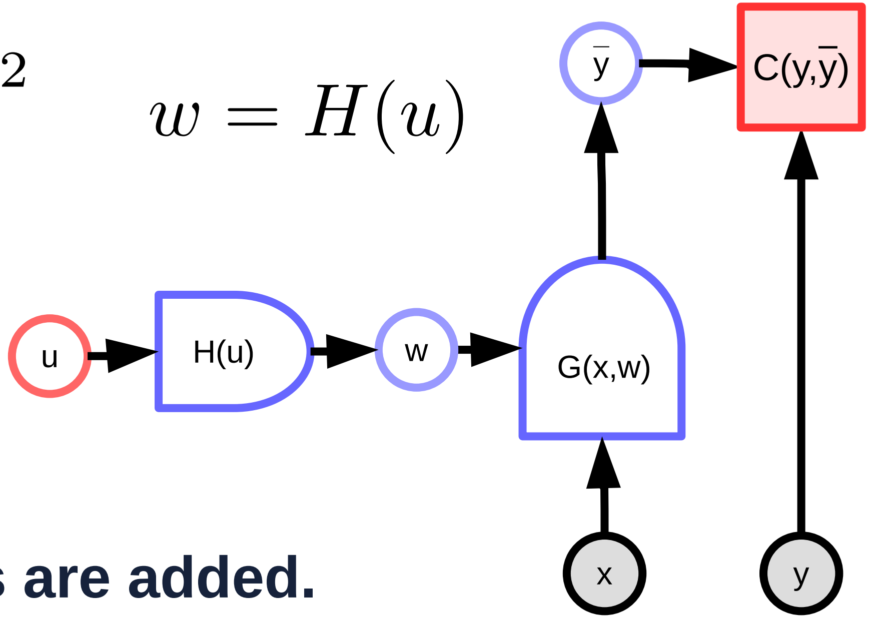


- **When the parameter vector is the output of another network H(x,u)**
- **The weights of network G(x,w) are dynamically configured by network H(x,u)**
- **The concept is very powerful**
  - The idea is very old

Simple parameter transform: weight sharing

- **Function H(u) replicates one component of u into multiple components of w**
  - $w_1, = w_2 = u_1 \quad w_3 = w_4 = u_2$

$$w = H(u)$$

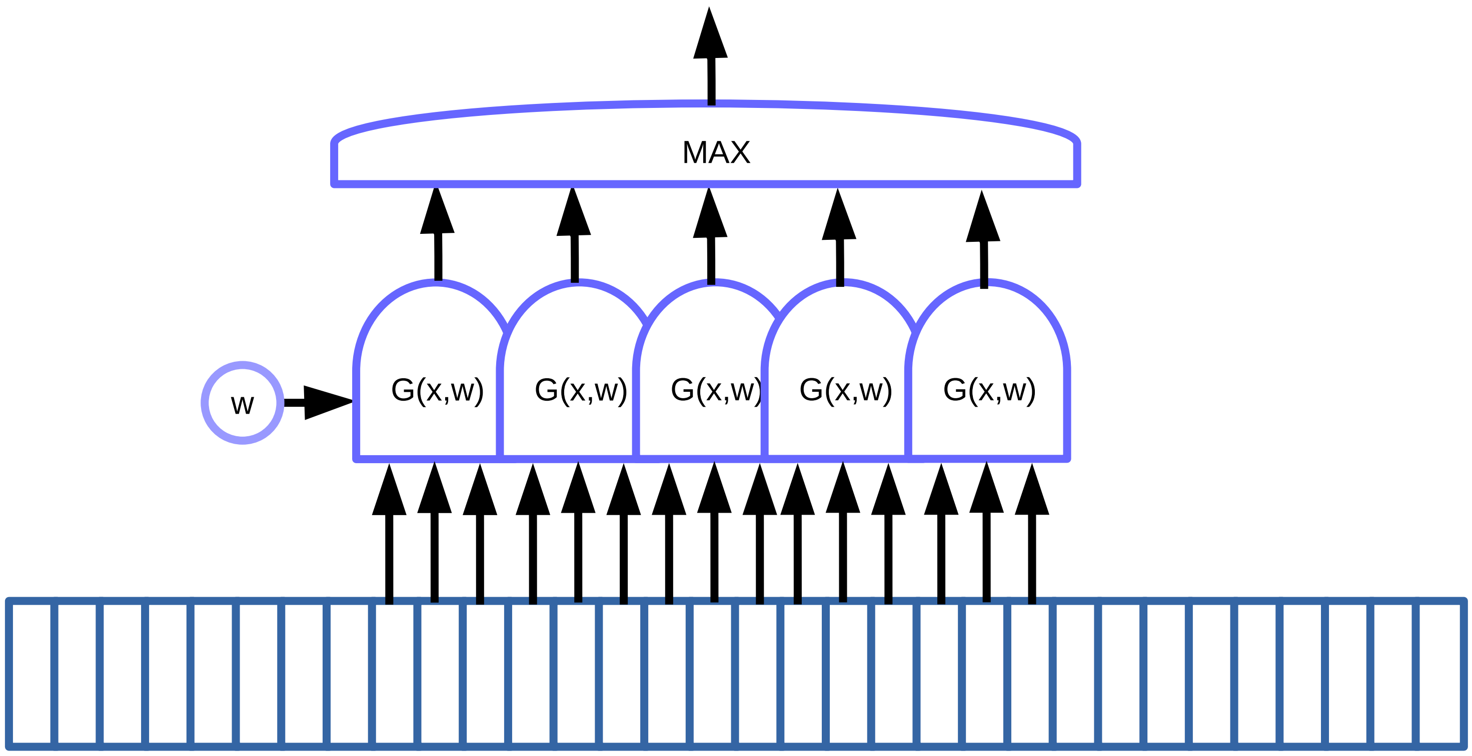


- **H is like a "Y" branch.**
  - Gradients are summed in the backprop

- **The gradients w.r.t. shared parameters are added.**

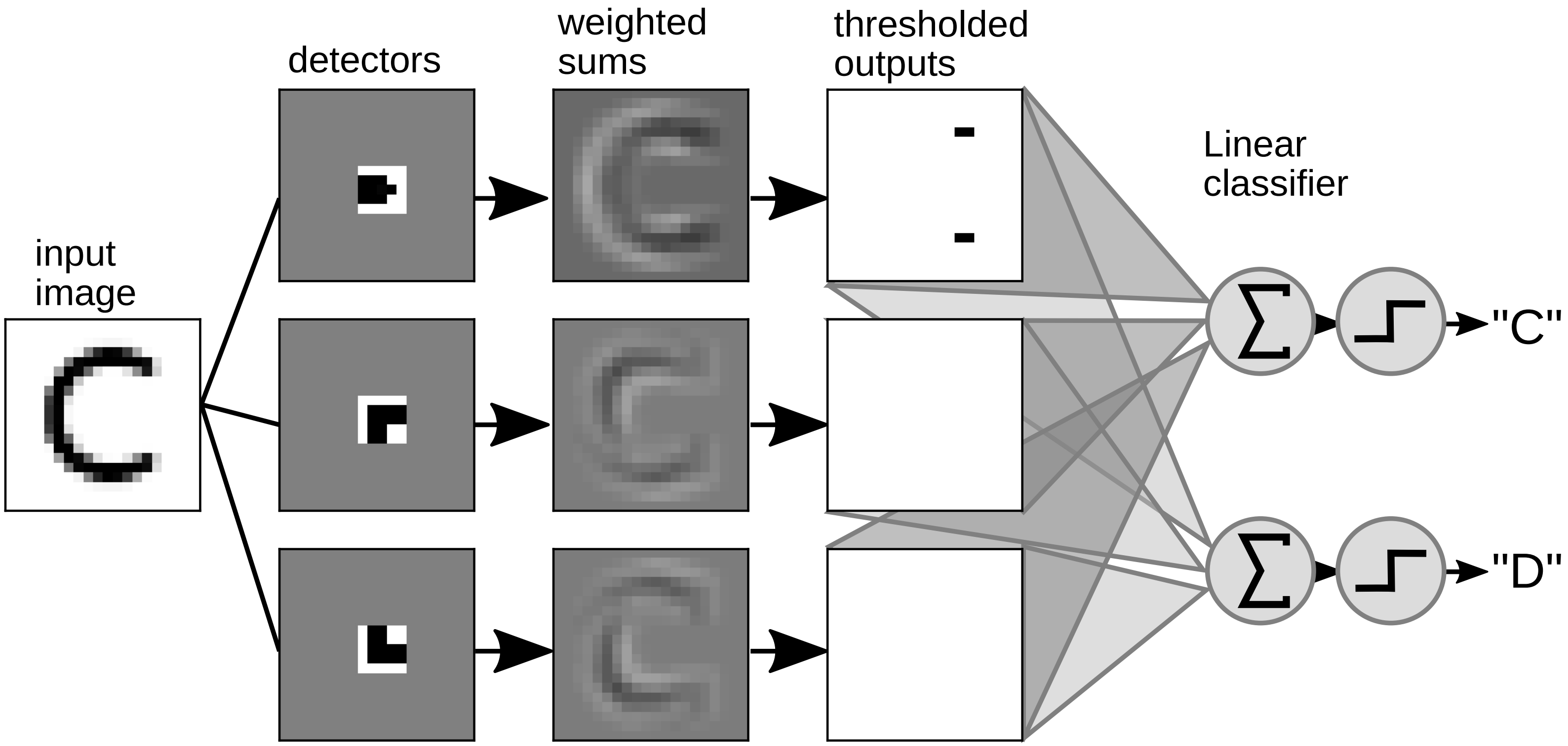
Shared Weights for Motif Detection

- **Detecting motifs anywhere on an input**



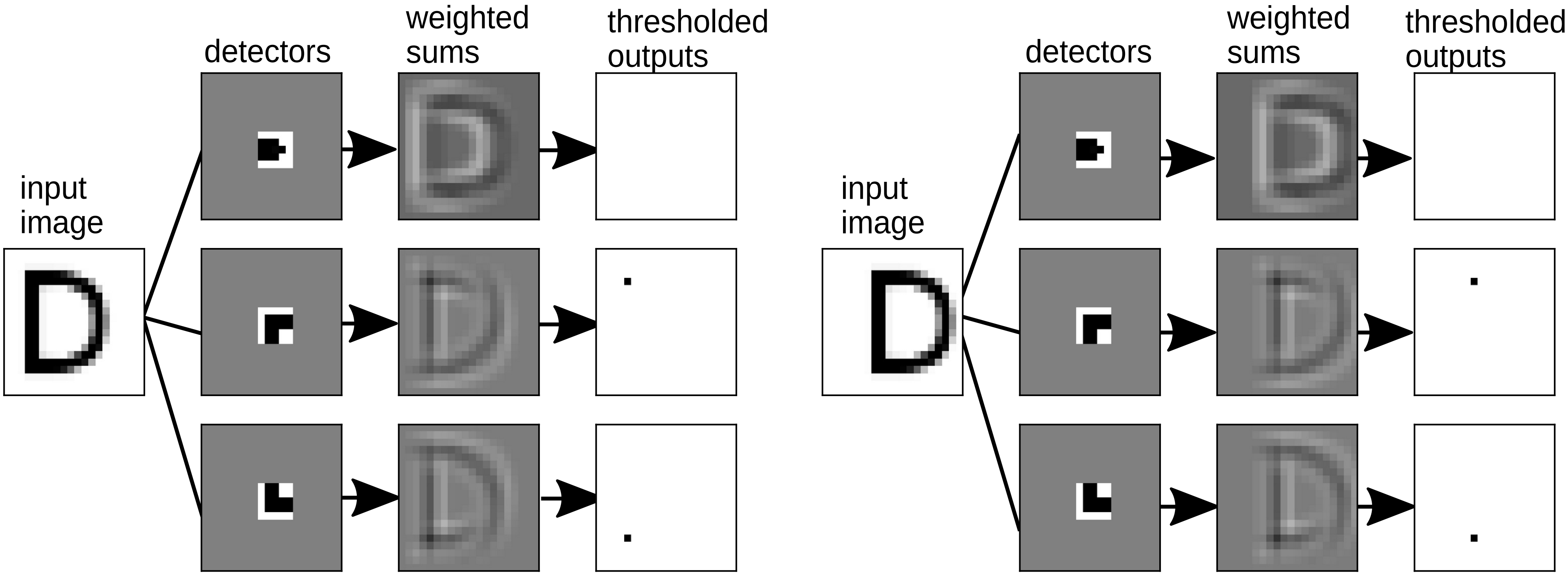
Detecting Motifs in Images

- **Swipe "templates" over the image to detect motifs**



Detecting Motifs in Images

- **Shift invariance**



Discrete Convolution (or cross-correlation)

- **Definition**
  - convolution

$$y_i = \sum_j w_j x_{i-j}$$

- **In practice**
  - Cross-correlation

$$y_i = \sum_j w_j x_{i+j}$$

If we read w backwards, cross-correlation ends up to the equivalent to convolution.
Mathmatically, convolution notation is more convenient for certain properties to have consistent form.
Programatically, w and x move on the same direction is more intuitive. DL frameworks choose the programatic way.

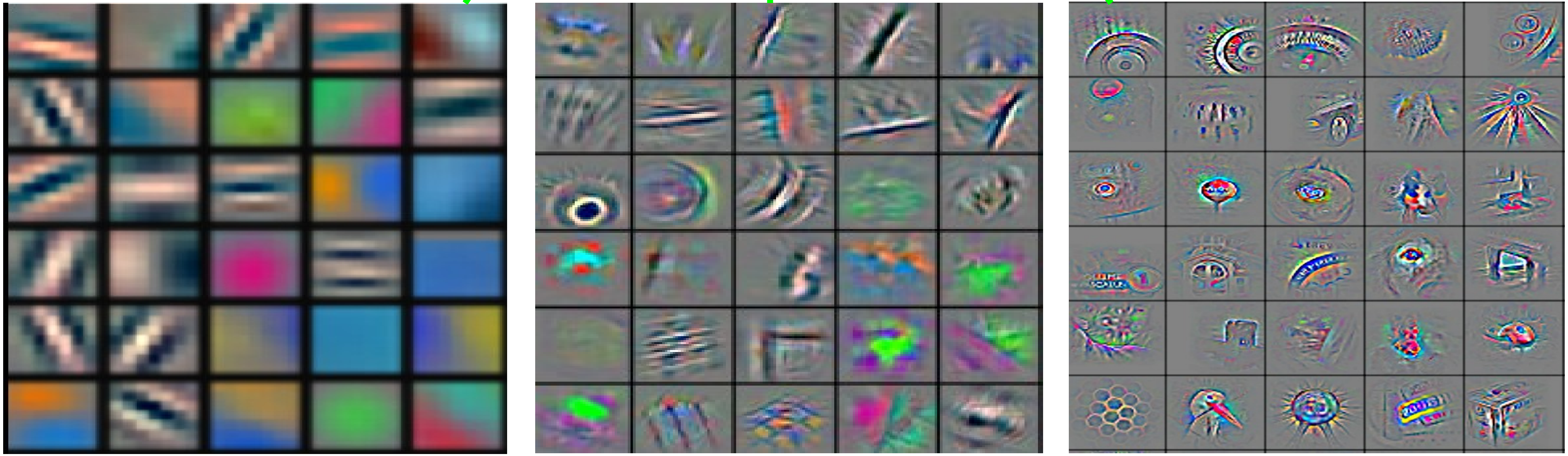
- **In 2D**

$$y_{ij} = \sum_{kl} w_{kl} x_{i+k,j+l}$$

Deep Learning = Learning Hierarchical Representations

- It's **deep** if it has **more than one stage** of non-linear feature transformation

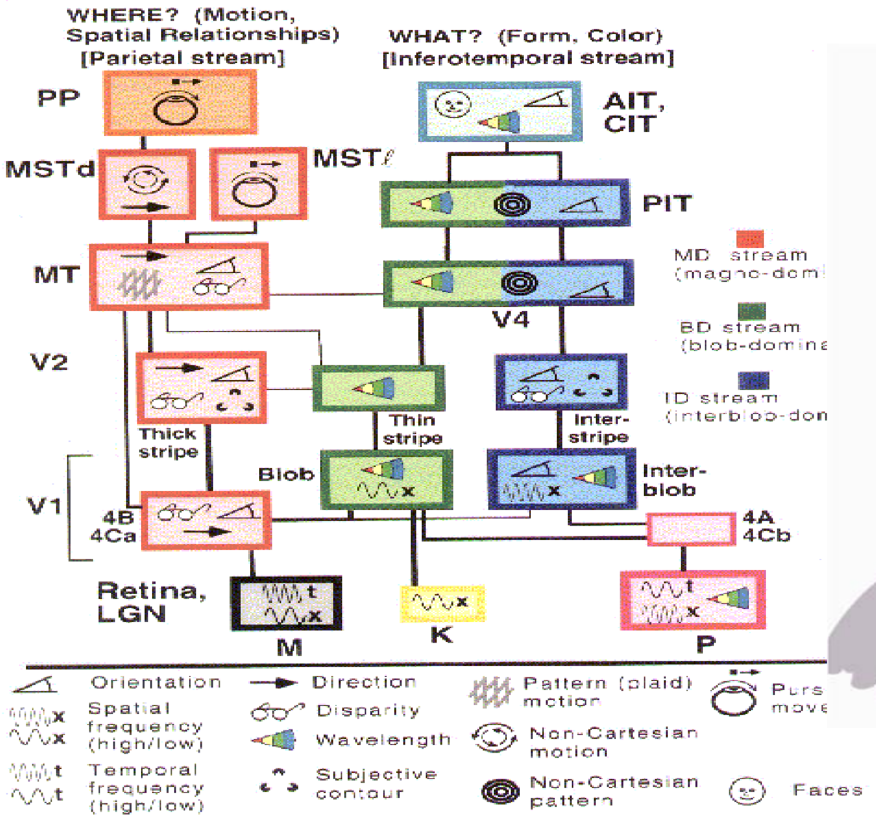
Low-Level Feature → Mid-Level Feature → High-Level Feature → Trainable Classifier →

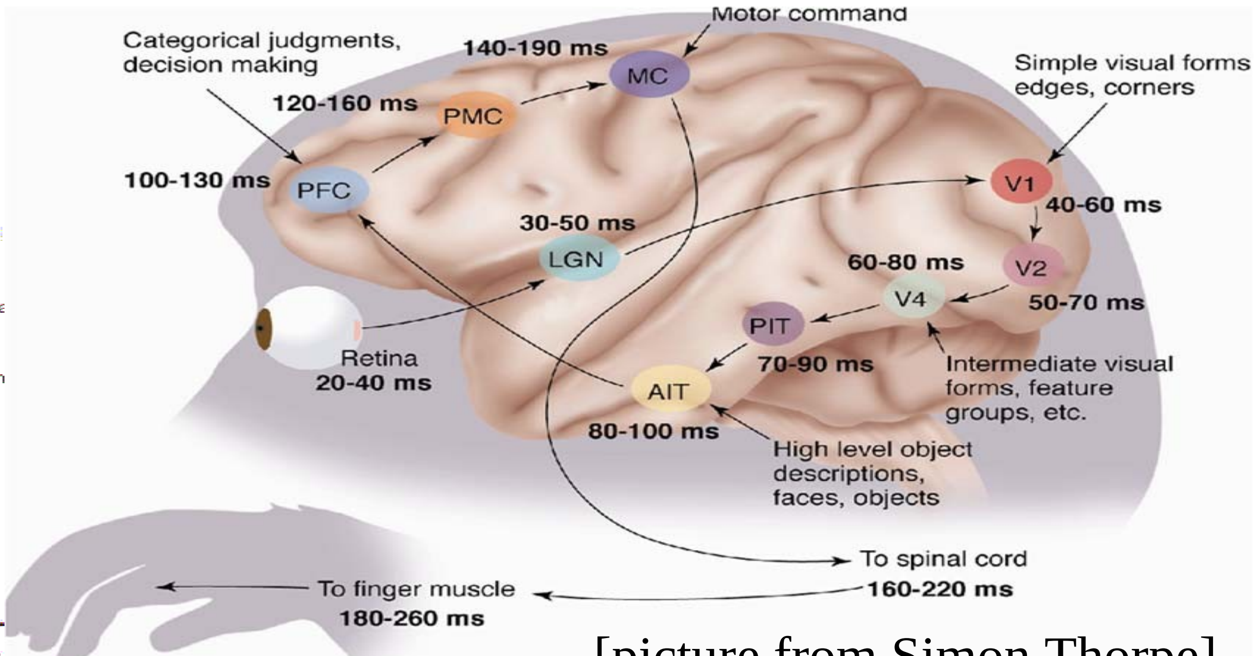


Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

How does the brain interprets images?

- **The ventral (recognition) pathway in the visual cortex has multiple stages**
- **Retina - LGN - V1 - V2 - V4 - PIT - AIT ....**



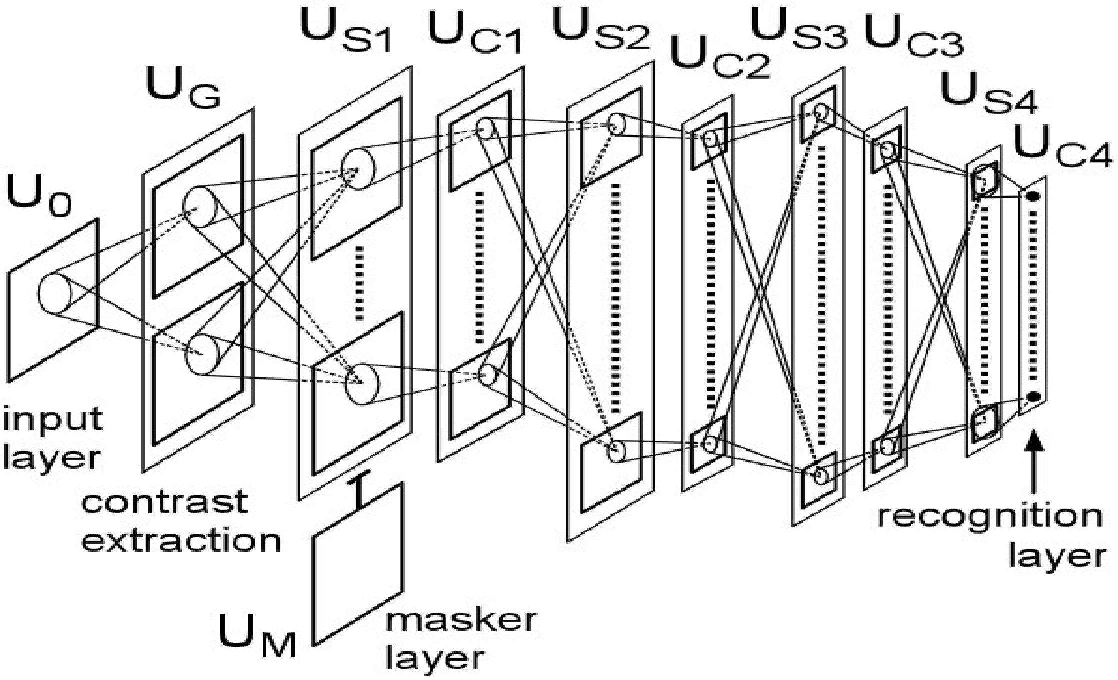


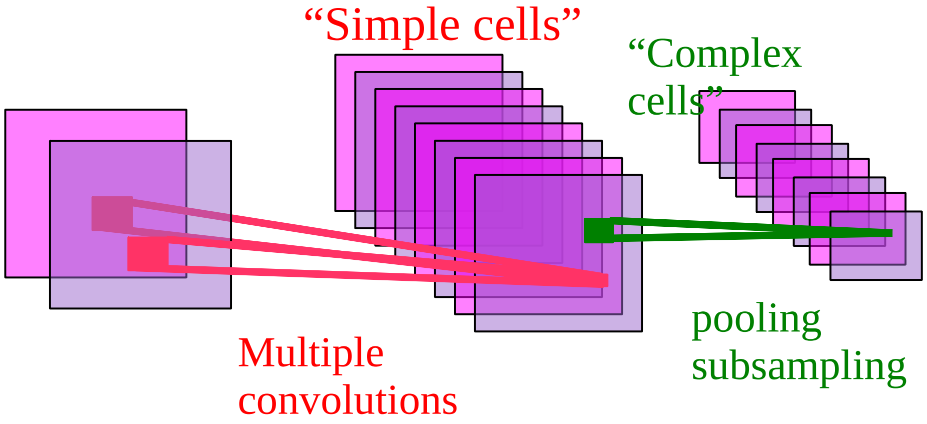
[picture from Simon Thorpe]

[Gallant & Van Essen]

Hubel & Wiesel's Model of the Architecture of the Visual Cortex

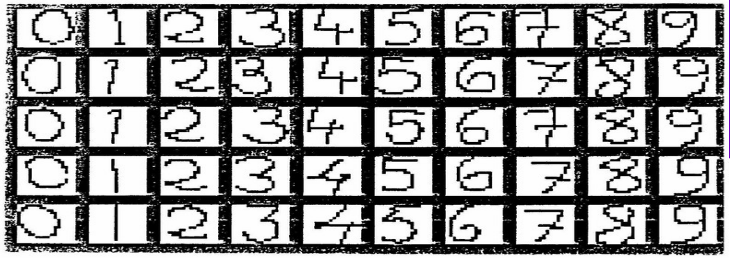
- **[Hubel & Wiesel 1962]:**
  - simple cells detect local features
  - complex cells "pool" the outputs of simple
  d.



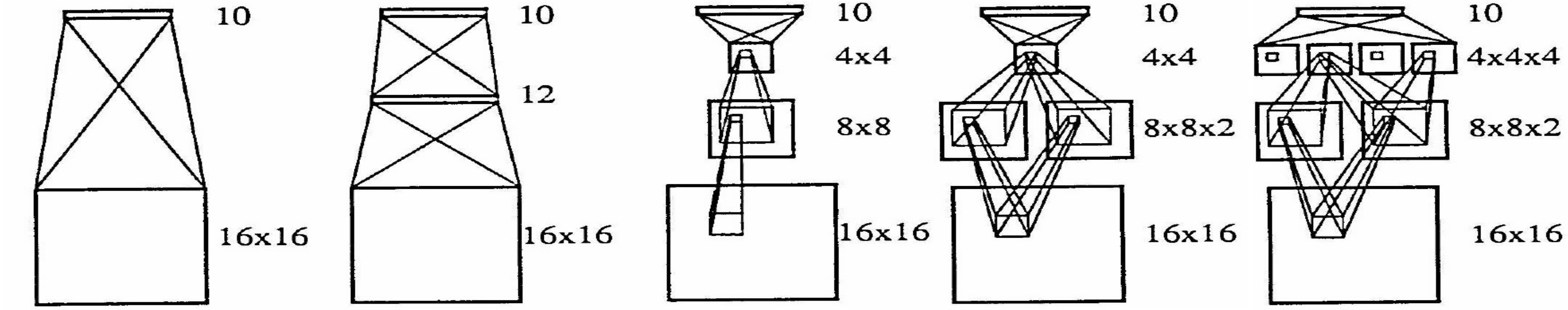


[Fukushima 1982][LeCun 1989, 1998],[Riesenhuber 1999]......

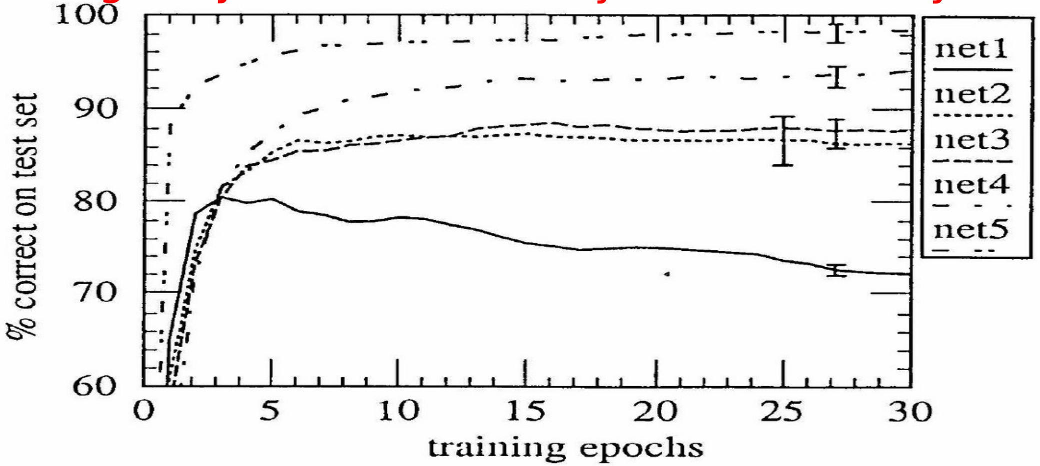
First ConvNets
(U Toronto)[LeCun 88, 89]



- **Trained with Backprop. 320 examples.**



Single layer   Two layers FC   locally connected   Shared weights   Shared weights



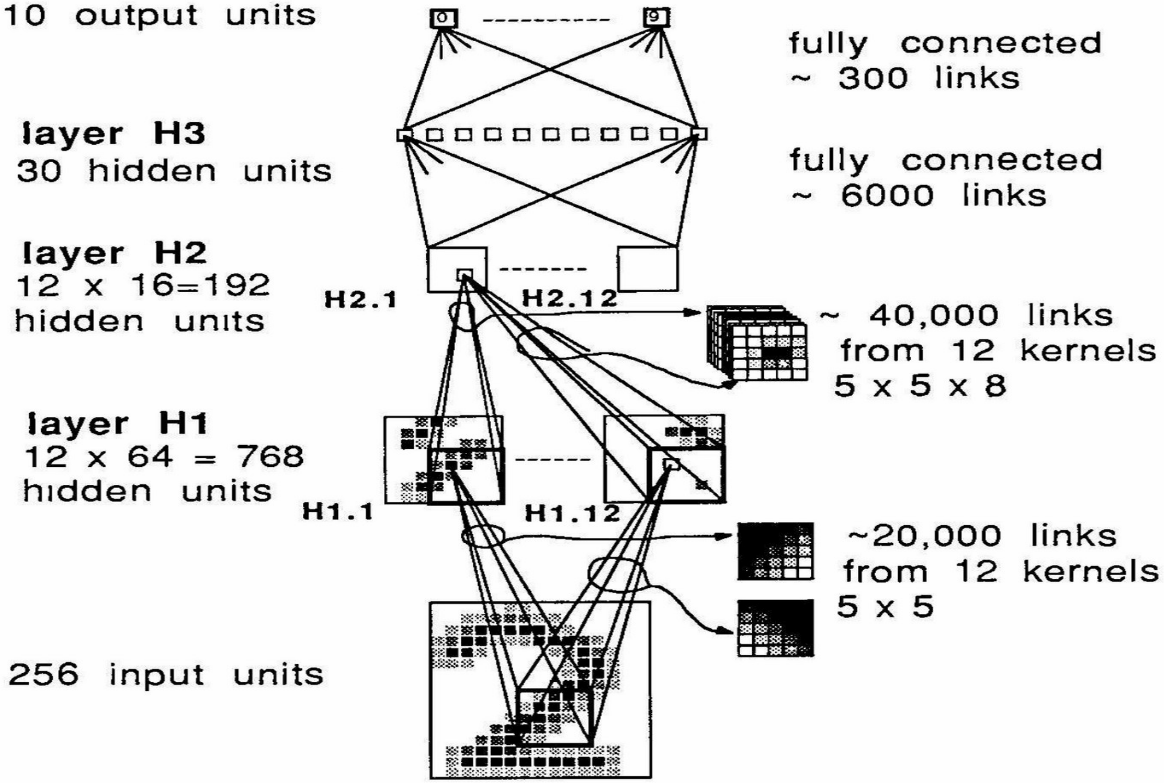
- Convolutions with stride (subsampling)

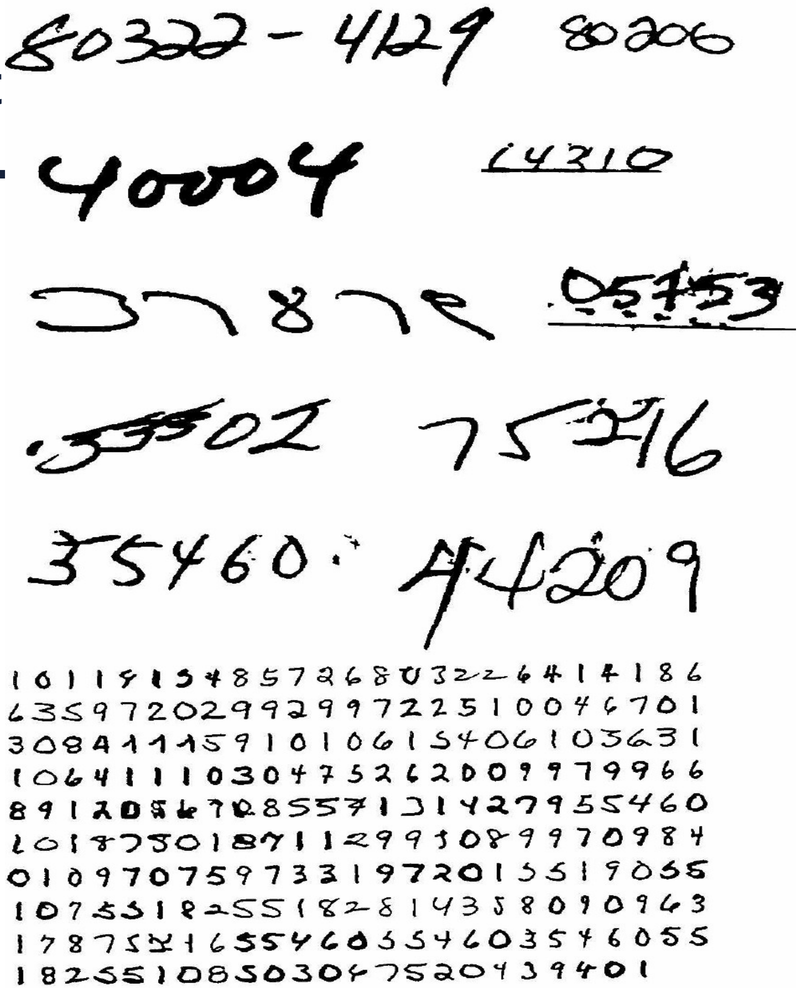
- No separate pooling layers

network architecture	links	weights	performance
single layer network	2570	2570	80 %
two layer network	3240	3240	87 %
locally connected	1226	1226	88.5 %
constrained network	2266	1132	94 %
constrained network 2	5194	1060	98.4 %

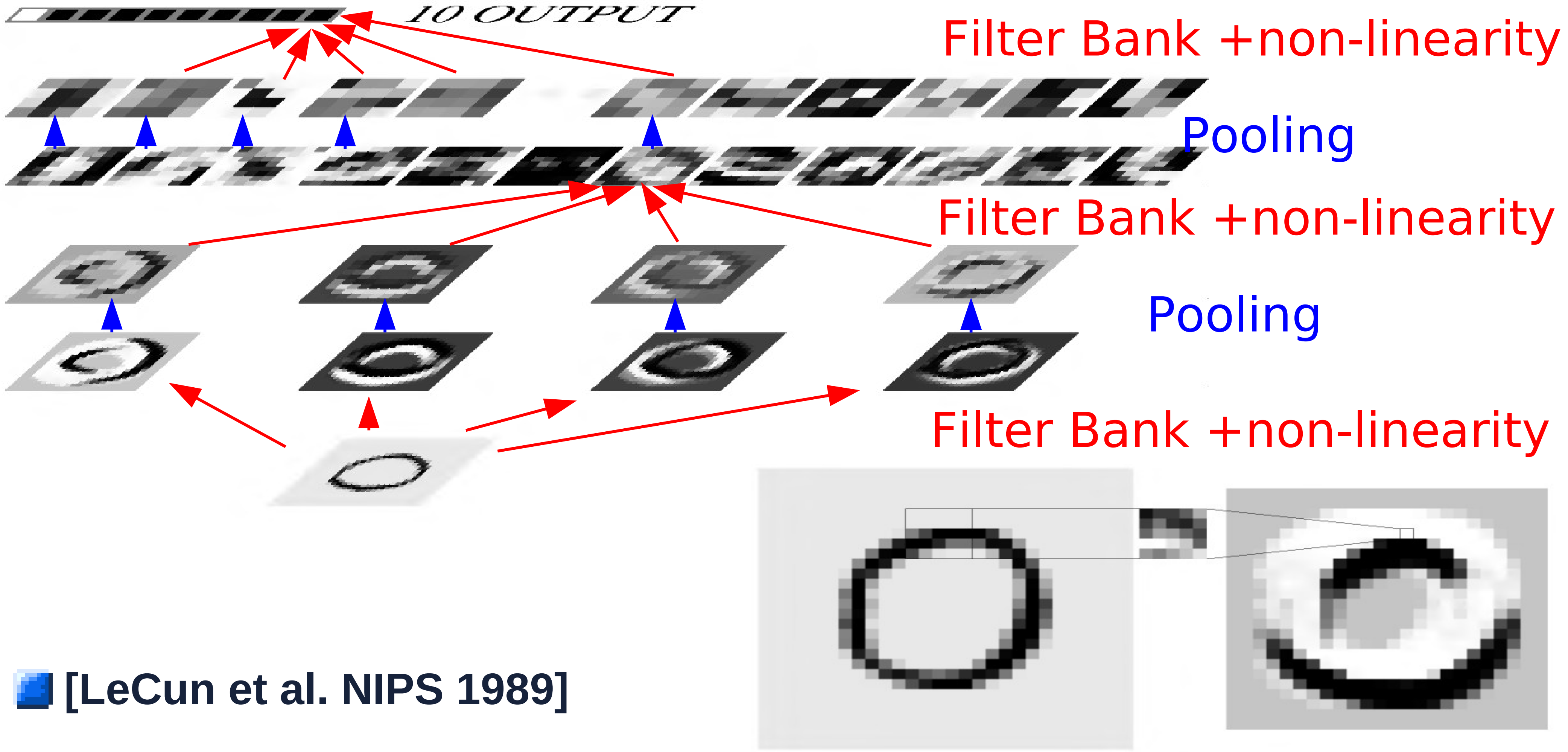
First “Real” ConvNets at Bell Labs [LeCun et al 89]

- **Trained with Backprop.**
- **USPS Zipcode digits: 7300 training, 2000 test**
- **Convolution with stride. No separate pooling.**





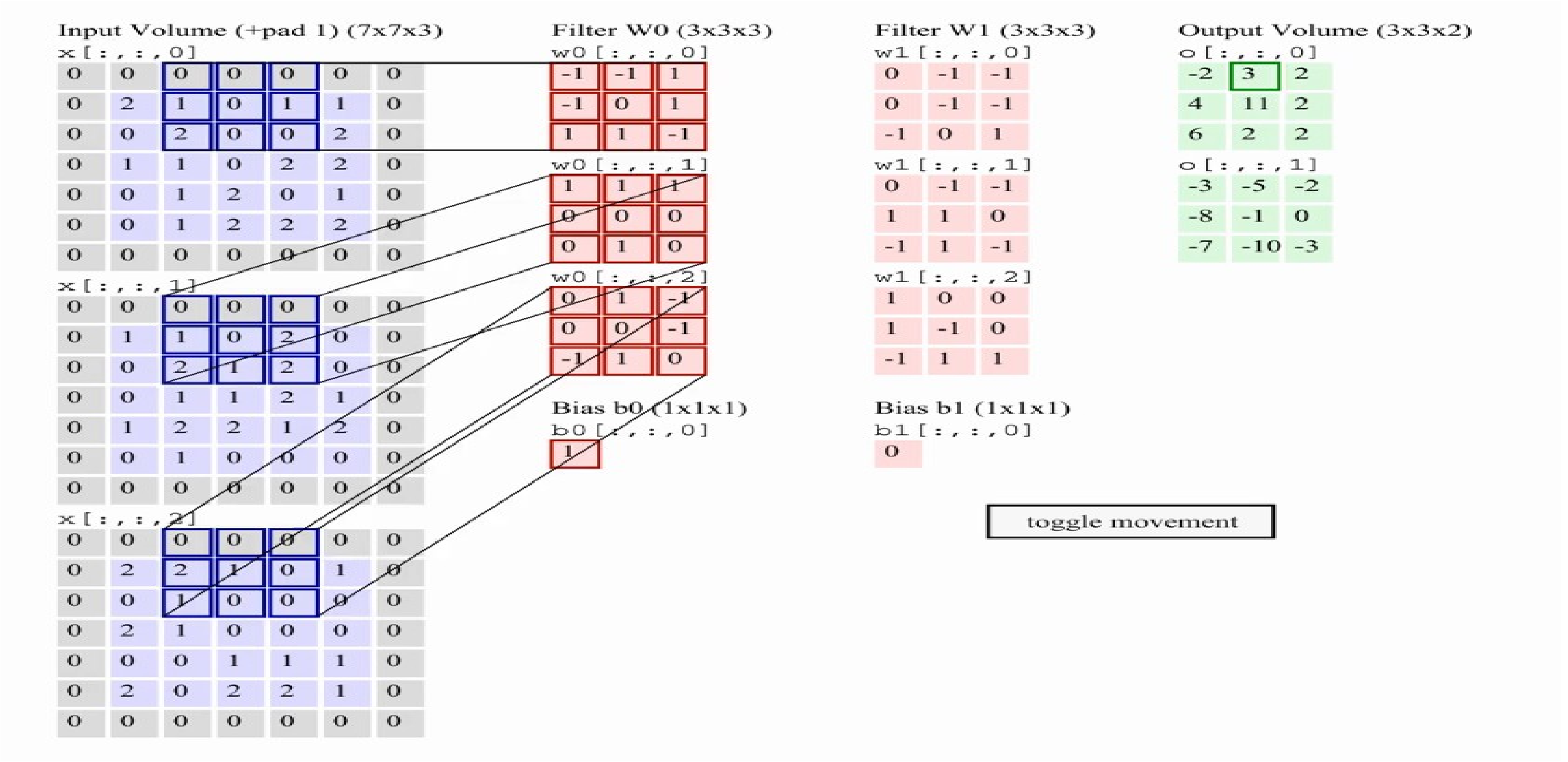
Convolutional Network Architecture



- [LeCun et al. NIPS 1989]

Y. LeCun

Multiple Convolutions

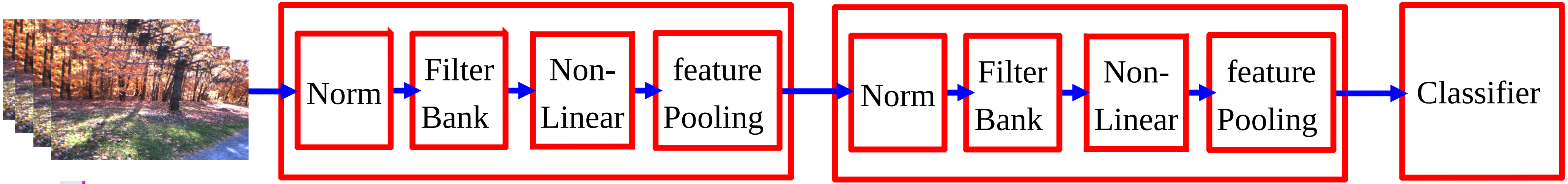


Animation: Andrej Karpathy http://cs231n.github.io/convolutional-networks/

Convolutional Network (vintage 1990)

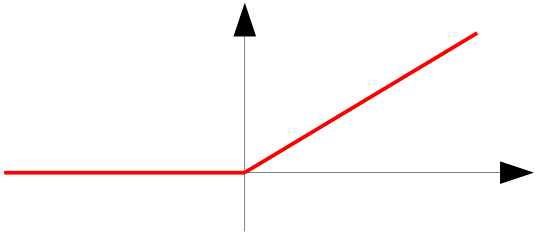
- **Filters-tanh → pooling → filters-tanh → pooling → filters-tanh**

Overall Architecture: multiple stages of Normalization → Filter Bank → Non-Linearity → Pooling



- **Normalization: variation on whitening (optional)**
  - Subtractive: average removal, high pass filtering
  - Divisive: local contrast normalization, variance normalization
- **Filter Bank: dimension expansion, projection on overcomplete basis**
- **Non-Linearity: sparsification, saturation, lateral inhibition....**
  - Rectification (ReLU), Component-wise shrinkage, tanh,..

$$ReLU(x) = max(x, 0)$$



- **Pooling: aggregation over space or feature type**
  - Max, Lp norm, log prob.

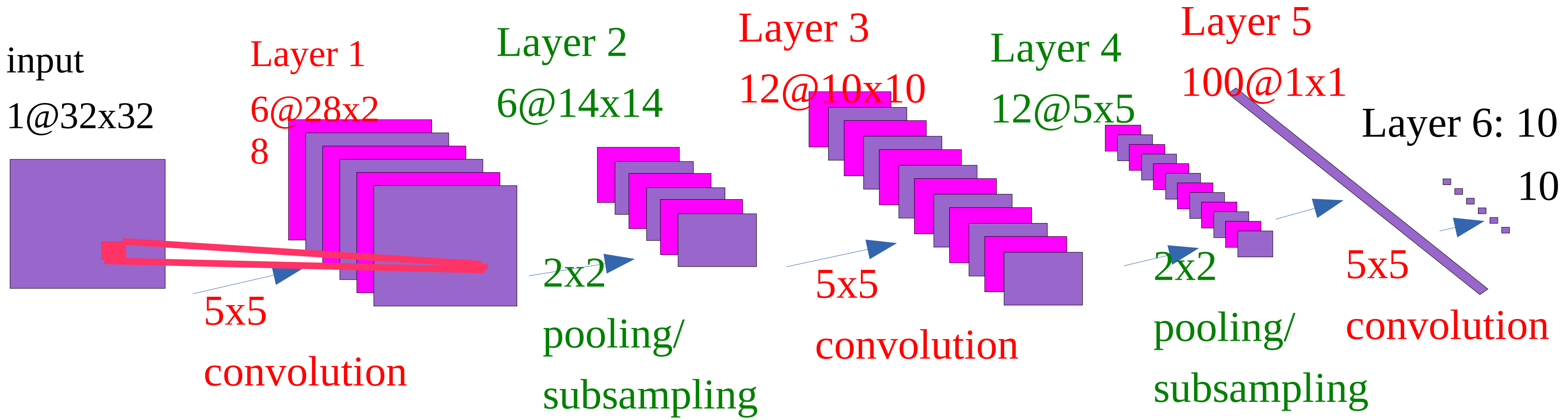
$$MAX: Max_i(X_i); \quad L_p: \sqrt[p]{X_i^p}; \quad PROB: \frac{1}{b}\log\left(\sum_i e^{bX_i}\right)$$

LeNet5

- **Simple ConvNet**
- **for MNIST**
- **[LeCun 1998]**

- **PyTorch code ------→**
  - (slightly different net)

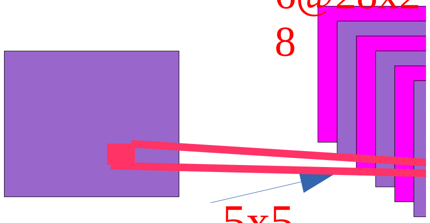
```python
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 20, 5, 1)
        self.conv2 = nn.Conv2d(20, 50, 5, 1)
        self.fc1 = nn.Linear(4*4*50, 500)
        self.fc2 = nn.Linear(500, 10)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        x = F.max_pool2d(x, 2, 2)
        x = F.relu(self.conv2(x))
        x = F.max_pool2d(x, 2, 2)
        x = x.view(-1, 4*4*50)
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        return F.log_softmax(x, dim=1)
```



LeNet5

- Simple ConvNet
- for MNIST
- [LeCun 1998]

- PyTorch code --→
  github.com/activatedgeek/LeNet-5
- nn.sequential() with ordered dictionary argument.

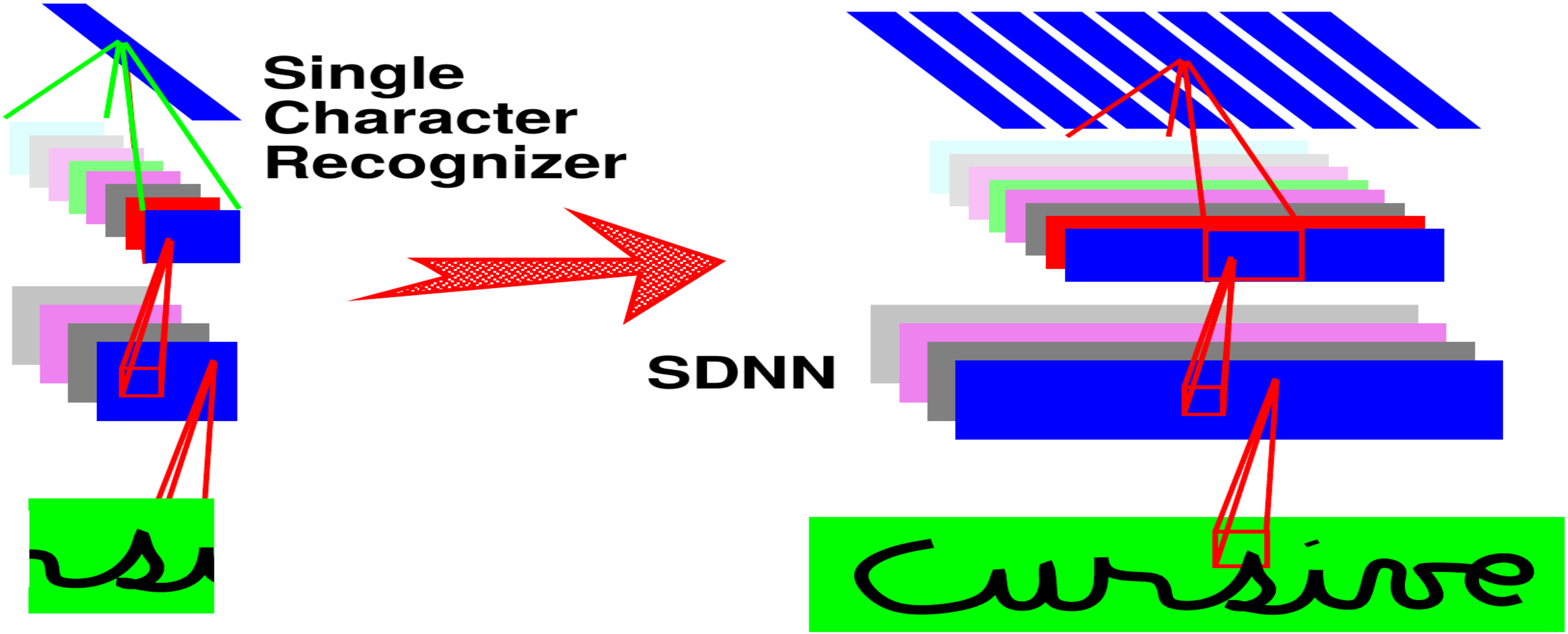



```python
    def __init__(self):
        super(LeNet5, self).__init__()

        self.convnet = nn.Sequential(OrderedDict([
            ('c1', nn.Conv2d(1, 6, kernel_size=(5, 5))),
            ('relu1', nn.ReLU()),
            ('s2', nn.MaxPool2d(kernel_size=(2, 2), stride=2)),
            ('c3', nn.Conv2d(6, 16, kernel_size=(5, 5))),
            ('relu3', nn.ReLU()),
            ('s4', nn.MaxPool2d(kernel_size=(2, 2), stride=2)),
            ('c5', nn.Conv2d(16, 120, kernel_size=(5, 5))),
            ('relu5', nn.ReLU())
        ]))

        self.fc = nn.Sequential(OrderedDict([
            ('f6', nn.Linear(120, 84)),
            ('relu6', nn.ReLU()),
            ('f7', nn.Linear(84, 10)),
            ('sig7', nn.LogSoftmax(dim=-1))
        ]))

    def forward(self, img):
        output = self.convnet(img)
        output = output.view(img.size(0), -1)
        output = self.fc(output)
        return output
```

Multiple Character Recognition [Matan et al 1992]

- **Every layer is a convolution**

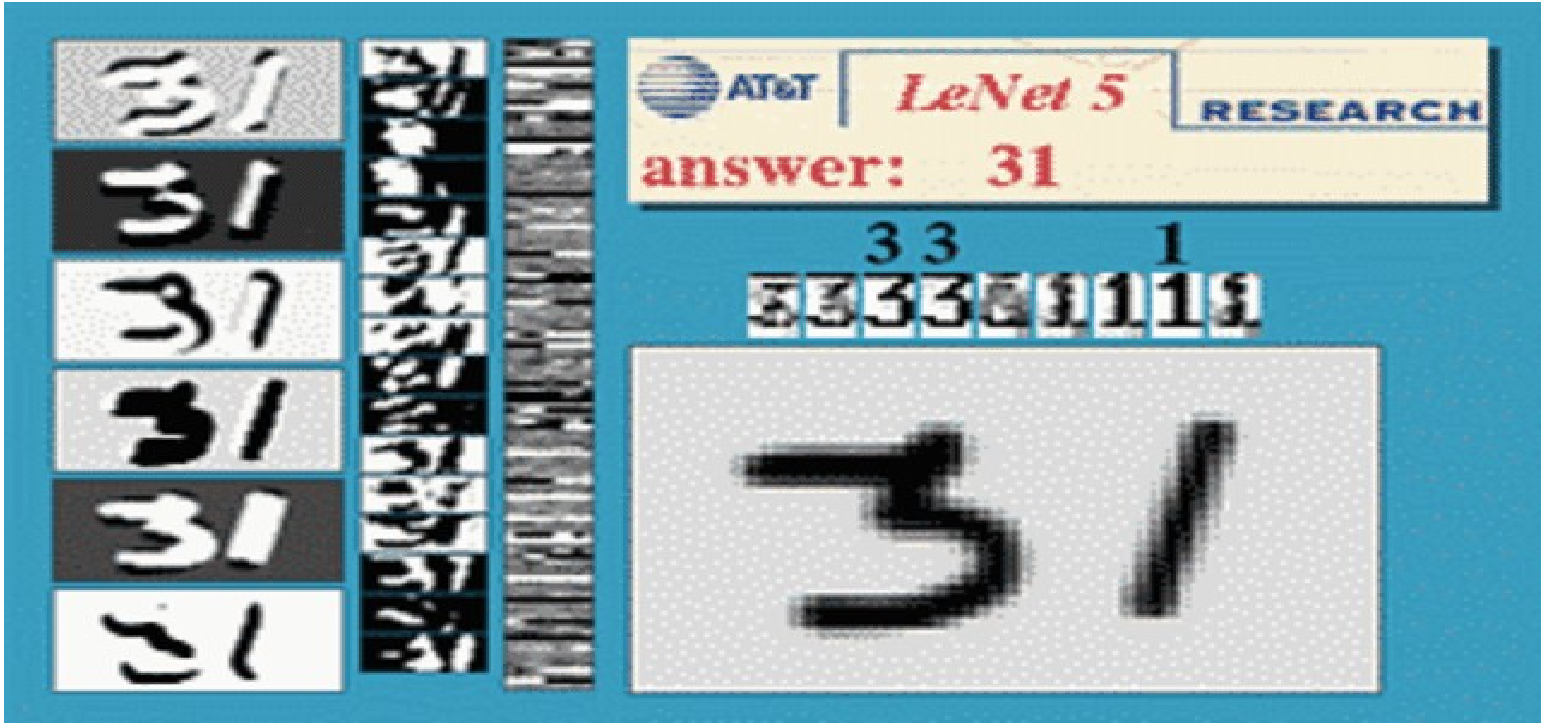
This is much cheaper than fixed size input and compute convolution again and again.
The last layer is 1x1 convolution



Sliding Window ConvNet + Weighted Finite-State Machine

Y. LeCun

Sliding Window ConvNet + Weighted FSM



What are ConvNets Good For

- **Signals that comes to you in the form of (multidimensional) arrays.**
- **Signals that have strong local correlations**
- **Signals where features can appear anywhere**
- **Signals in which objects are invariant to translations and distortions.**
- **1D ConvNets: sequential signals, text**
  - Text, music, audio, speech, time series.
- **2D ConvNets: images, time-frequency representations (speech and audio)**
  - Object detection, localization, recognition
- **3D ConvNets: video, volumetric images, tomography images**
  - Video recognition / understanding
  - Biomedical image analysis
  - Hyperspectral image analysis